МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»
(НИЯУ МИФИ)

ИНСТИТУТ ЯДЕРНОЙ ФИЗИКИ И ТЕХНОЛОГИЙ
КАФЕДРА №40 «ФИЗИКА ЭЛЕМЕНТАРНЫХ ЧАСТИЦ»

ОТЧЕТ
О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

ПРОГРАММНАЯ ПЛАТФОРМА ДЛЯ МНОГОЭТАПНОЙ ОБРАБОТКИ ДАННЫХ ФИЗИЧЕСКОГО ЭКСПЕРИМЕНТА НА ОСНОВЕ ФРЕЙМВОРКА GAUDI

Научный руководитель
доц. к.ф-м.н. _____ Е. Ю. Солдатов

Научный консультант
к.ф-м.н. _____ А. С. Жемчугов

Студент _____ Л. Л. Симбирятин

Москва 2025

Содержание

Введение

Ускорительный комплекс NICA (Nuclotron based Ion Collider fAcility) является проектом масштаба мегасайенс, реализуемым на базе ОИЯИ (Дубна, Россия). На коллайдере предусмотрены две точки пересечения пучков заряженных частиц, в одной из которых предполагается установить детектор SPD (Spin Physics Detector) с целью изучения спиновой структуры протона и дейтрона. Коллайдер предоставляет уникальную возможность для изучения поляризованных pp и dd столкновений с $\sqrt{s} = 27$ ГэВ и светимостью порядка 10^{32} см$^{-2}$с$^{-1}$.

Рисунок 1 — Ускорительный комплекс NICA

Как и любой крупный эксперимент, SDP предполагает разработку своего физического ПО. Такое ПО, главным образом, предназначено для реконструкции событий, генерации Монте-Карло наборов, а также для автономной обработки данных. Для каждой из перечисленных задач существуют специализированные библиотеки. Интеграция этих библиотек в общий фреймворк является важной задачей.

Текущим вариантом физического ПО эксперимента SPD является пакет SpdRoot, написанный на основе фреймворка FairRoot [1]. С его помощью рассчитывают физические показатели детектора, производят возможные оптимизации и прочие подготовительные мероприятия. Однако, являясь непосредственным наследником пакета Root, SpdRoot наследует, в том числе, и все его недостатки. Также в SpdRoot не поддерживаются методы многопоточного программирования.

По этим причинам к началу набора данных необходимо разработать фреймворк для физического ПО эксперимента SPD на базе Gaudi [2]. В рамках прошедшего семестра были выполнены следующие задачи:

- интеграция библиотеки Pythia8 (генерация первичных вершин);
- интеграция библиотеки GeoModel (описание геометрии детектора);
- начало интеграции пакета Geant4;

1 Эксперимент SPD

Несмотря на важность поиска проявлений частиц, выходящих за рамки Стандартной модели, в рамках барионной материи по-прежнему остается множество открытых вопросов. Даже протон не может считаться в полной мере изученной частицей. В наивной кварковой модели протон представляет собой комбинацию двух u и одного d кварка. Эта простейшая кварковая модель позволяет предсказать такие свойства как электрический заряд, изоспин, четность, магнитный момент. Этот результат является действительно удивительным, ведь такая модель не учитывает угловые моменты кварков, морские кварки, а также глюоны. КХД является современным инструментом описания сильного взаимодействия, она с успехом применяется для описания множества процессов. Основным нюансом КХД является ее непертурбативность на низких энергиях, в частности, одной из нерешенных проблем остается описание свойств адронов (в том числе и протона) напрямую из динамики составляющих их кварков и глюонов.

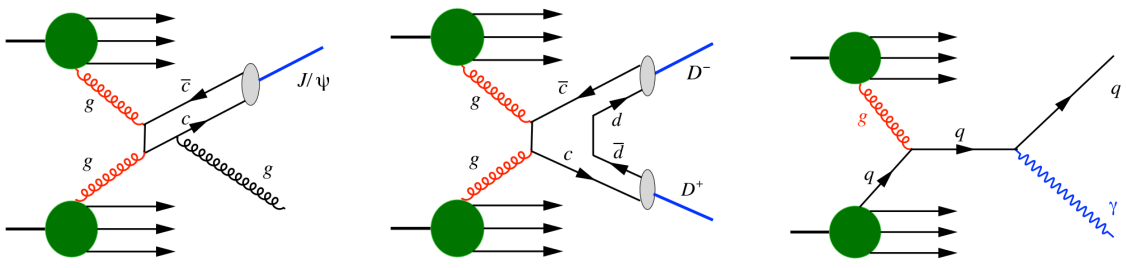
Детектор SPD (Spin Physics Detecror) будет размещен в одной из двух точек столкновения пучков коллайдера NICA (Дубна, Россия). Целью построения SPD является изучение спиновой структуры нуклонов в поляризованных pp ($\sqrt{s} < 27$ ГэВ) и dd ($\sqrt{s} < 13.5$ ГэВ) столкновениях.

1.1 Основная цель эксперимента

Одним из способов описания внутренней партонной структуры нуклона является использование функций партонных распределений PDF (Parton Distribution Function). В неполяризованном простейшем случае эта функция описывает вероятность найти внутри нуклона партон, несущий определенную долю общего импульса. В общем же случае необходимо также учитывать не только продольную компоненту, но и поперечную (например, эффект Сиверса [3]), а также поляризацию как самого нуклона, так и партонов внутри него.

В то время как вклад кварков в общий спин нуклона был довольно точно измерен коллаборациями EMC, HERMES и COMPASS, измерения по глюонной компоненте либо являются менее точными, либо отсутствуют вовсе.

Основная цель эксперимента SPD - извлечь информацию о глюонных функциях распределения, зависящих от поперечного импульса (TMD PDFs), для протона и дейтрона, через измерение одинарных и двойных спиновых асимметрий в процессах рождения чармониев, очарованных частиц, а также прямых фотонов [4].



а б в

Рисунок 2 — Фейнмановские диаграммы процессов-пробников: рождение (а) чармониев, (б) очарованных частиц, (в) прямых фотонов

1.2 Детектор

Основной целью эксперимента является извлечение TMD PDF для глюонов через измерение спиновых асимметрий в процессах рождения чармониев, очарованных частиц, а также прямых фотонов. Поставленная цель, а также обозначенные процессы-пробники определяют вид и необходимые характеристики детектора SPD.

SPD представляет собой универсальный 4π детектор с характерной для коллайдерных экспериментов цилиндрической формой. Его компонентами являются:

- кремниевый вершинный детектор (VD) с разрешением выше 100 мкм для реконструкции вторичных вершин распадов D мезонов;
- трековая система (TS) $\sigma_{p_T}/p_T \approx 2\%$;
- время-пролетная система (TOF) с разрешением порядка 60 пс для разделения π/K и K/p;
- детектор FARICH для улучшения разделения π/K и K/p;
- электромагнитный калориметр (ECal) с энергетическим разрешением $\sim 5\%/\sqrt{E}$ для регистрации фотонов;
- мюонная система (RS);
- пара счетчиков столкновений (BBC) и калориметров нулевых углов (ZDC) для контроля поляризации и светимости;

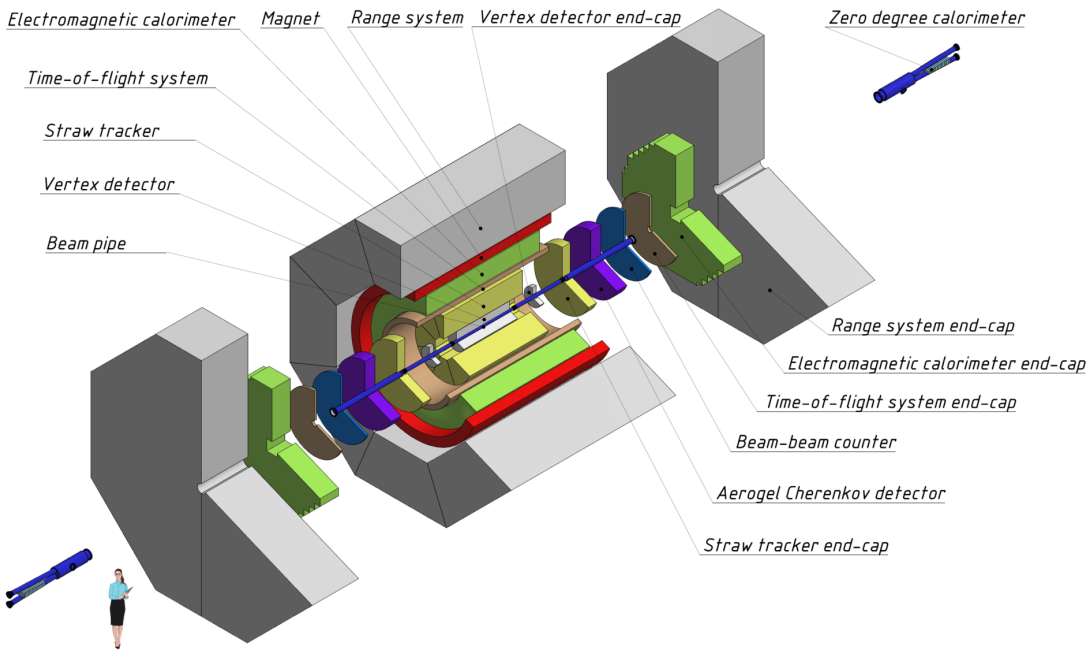


Рисунок 3 — Макет детектора SPD в полной сборке

В силу трудностей, возникающих при построении аппаратного триггера, для SPD предполагается безтриггерная система сбора данных. В совокупности с высокой частотой столкновений (до 12 МГц) и сотнями тысяч каналов детектора это представляет собой сложную задачу по разработке эффективной системы сбора и обработки данных.

2 Offline программное обеспечение эксперимента SPD

Offline программное обеспечение предназначено для решения таких задач, как реконструкция событий, их моделирование, а также проведение физического анализа полученных в результате эксперимента данных. Схематично эти этапы жизненного цикла данных представлены на рисунке 4. Все эти этапы в рамках фреймворка должны быть объединены в единую инфраструктуру.

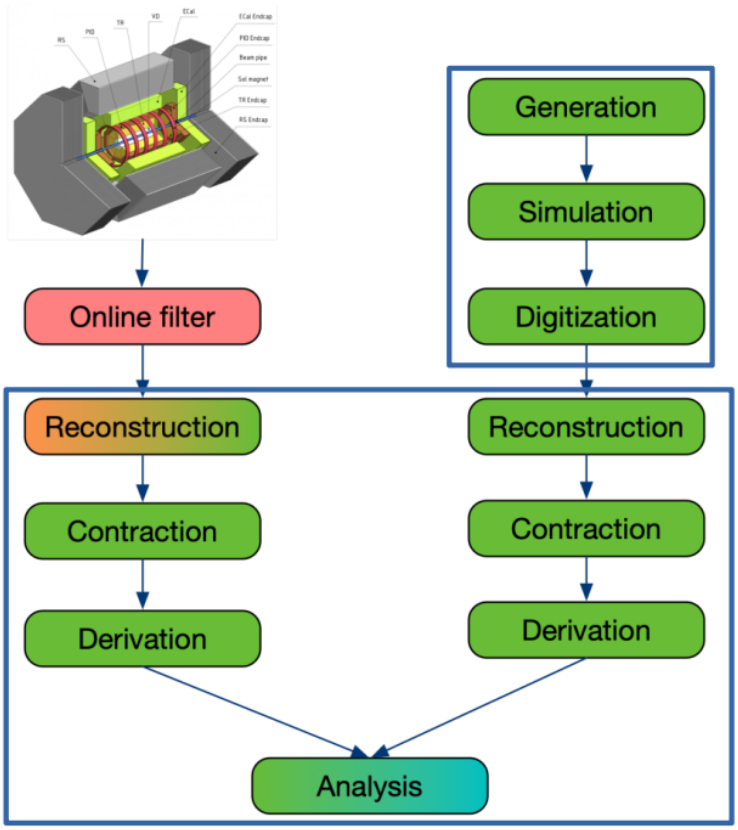


Рисунок 4 — Этапы обработки данных. За выделенные этапы отвечает offline ПО

2.1 Генерация и моделирование

Для моделирования протон-протонных столкновений используется генератор Pythia8 [5], дейтрон-дейтронные столкновения моделируются с помощью модели FRITIOF [6], а для моделирования ядро-ядерных столкновений используется генератор UrQMD [7]. Для моделирования распространения частиц в детекторе, а также формирования откликов в чувствительных элементах установки используется пакет Geant4 [8].

2.2 Описание детектора

Для описания геометрии детектора используется библиотека GeoModel [9]. Она позволяет хранить геометрию детектора как отдельный независимый компонент, в том числе записывать ее в базу данных SQLite. Выделение геометрии детектора в отдельный компонент обусловлено тем, что многие алгоритмы реконструкции должны взаимодействовать с ней. Геометрия внутри Geant4 будет создаваться путем конвертации геометрии из GeoModel.

3 Фреймворк Gaudi

Gaudi [2] представляет собой программный пакет, содержащий все необходимые интерфейсы и компоненты для написания на его основе фреймворков для экспериментов в области физики высоких энергий. Изначально Gaudi разрабатывался по внутренним нуждам коллаборации LHCb, однако вскоре после подключения к разработке коллаборации ATLAS стало ясно, что пакет может быть легко трансформирован и под любой другой эксперимент. Надежность пакета подтверждается его использованием в многочисленных коллаборациях по всему миру.

3.1 Архитектура Gaudi

Одним из принципиальных решений при создании Gaudi стала изоляция пользователя от деталей внутреннего устройства фреймворка. Достигается такая изоляция за счет построения архитектуры, представляющей собой набор компонентов и правил их взаимодействия. У каждого компонента есть свой интерфейс и функционал. Задача же пользователя сводится к доопределению функционала конкретного компонента с сохранением его интерфейса. Программно это осуществляется путем наследования от одного из базовых классов.

Другим принципиальным решением стало явное разделение между данными и алгоритмами, оперирующими этими данными. Такое разделение обусловлено естественным подходом: данные - это набор чисел, который не стоит перегружать каким-либо дополнительным функционалом, а алгоритмы - это математические процедуры, проводимые с этими числами. Также для хранения данных на диске необходимо предоставить соответствующие конвертеры. Таким образом, в Gaudi представлены следующие базовые классы, предназначенные для пользователя:

- DataObject
- Algorithm
- Converter

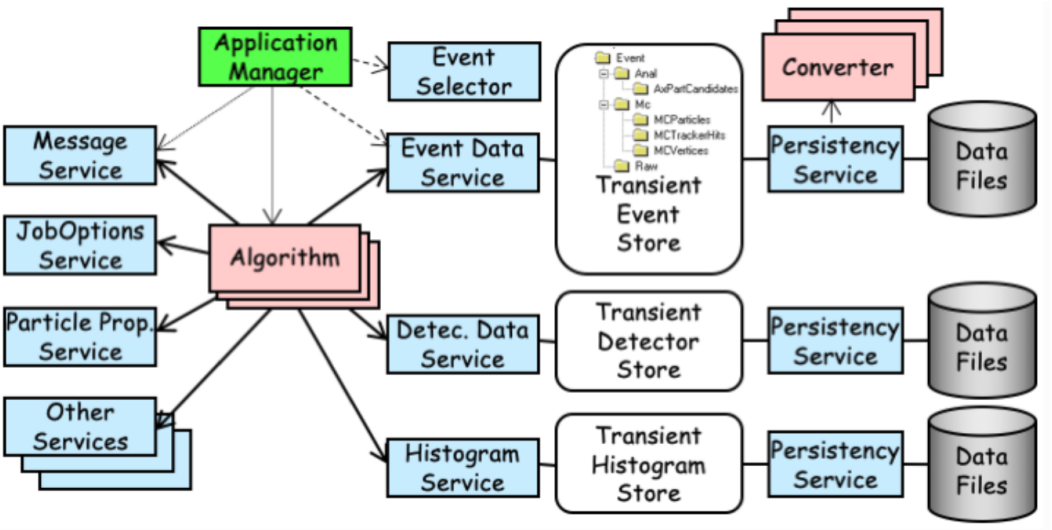


Рисунок 5 — Основные компоненты архитектуры Gaudi

3.2 Алгоритмы

Алгоритмы главным образом производят определенные действия с данными (генерация, реконструкция и т. п.), основная часть модификации фреймворка под нужды конкретного эксперимента заключается в написании алгоритмов.

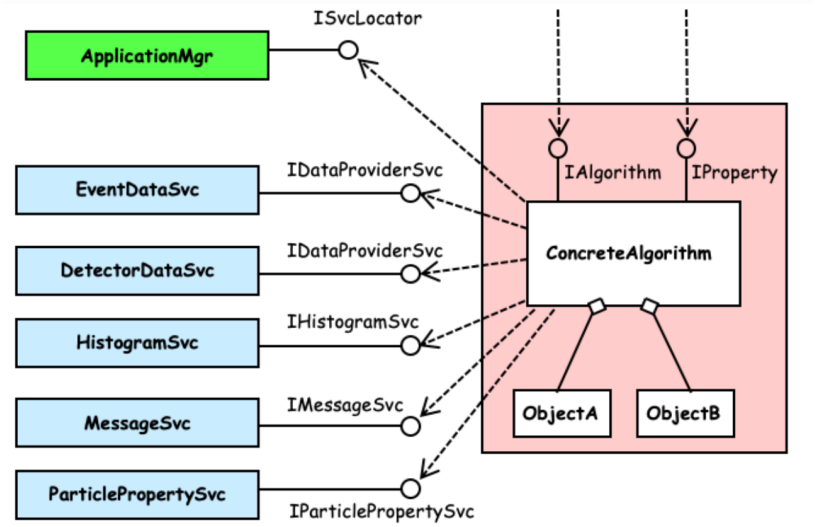


Рисунок 6 — Взаимодействие алгоритма с фреймворком в процессе работы

Алгоритм взаимодействует с элементами фреймворка посредством их интерфейсов. Например, доступ к данным и их запись во временное хранилище осуществляется посредством интерфейса IDataProviderSvc, а в случае необходимости вывода алгоритмом какой-либо информации можно воспользоваться MessageSvc, обращение к которому реализуется через интерфейс IMessageSvc.

Алгоритм является конфигурируемым. Так, перед запуском можно установить значения для внутренних переменных (например, пороговые значения для отборов событий). Это возможно благодаря тому, что базовый класс Algorithm реализует сразу два интерфейса, в том числе IProperty, что дает возможность сервису JobOptionSvc в момент конфигурирования обращаться к полям алгоритма и задавать их значения. Вторым интерфейсом, который реализуется базовым классом Algorithm, является IAlgorithm. IAlgorithm используется для управления алгоритмом в процессе работы фреймворка. Также этот интерфейс содержит три чисто виртуальных метода, реализация которых целиком ложится на конечного пользователя:

- **Initialize**, который может быть использован для создания выходных гистограмм, конфигурирования побочных алгоритмов и т.п.
- **Execute**, который вызывается единожды на событие и совершает соответствующие какой-либо физической задаче преобразования над данными, относящимися к этому событию. Для побочных алгоритмов execute можно вызывать более одного раза.
- **Finalize**, который вызывается в конце работы программы и может быть использован для подведения итоговой статистики, фитирования итоговых гистограмм и т.п.

3.3 Сервисы

Сервисы предназначены для решения общих задач, возникающих в ходе работы приложения. К таковым можно отнести чтение и запись данных в Transient Data Store, вывод сообщений, генерацию случайных чисел, получение свойств частиц и т. д. Сервисы не относятся к какому-то конкретному алгоритму, они наравне с алгоритмами являются самостоятельными компонентами фреймворка. Так, например, за создание конкретного набора алгоритмов на основе конфигурационного файла отвечает сервис JobOptionSvc.

Сервисы создаются единожды в начале работы приложения и затем вызываются другими компонентами фреймворка. При этом при создании используется ленивая инициализация. По умолчанию Application Manager создает только JobOptionsSvc и MessageSvc.

Обращение к сервису должно осуществляться через его интерфейс. Для того чтобы какой-либо компонент имел доступ к определенному сервису, компонент нужно снабдить ссылкой или указателем на этот сервис. Для этого внутри фреймворка существует функция serviceLocator.

Помимо создания алгоритмов, Gaudi также позволяет пользователю создавать собственные сервисы. Для этого необходимо предоставить интерфейс нового сервиса, также новый сервис должен быть наследником базового класса Sevice.

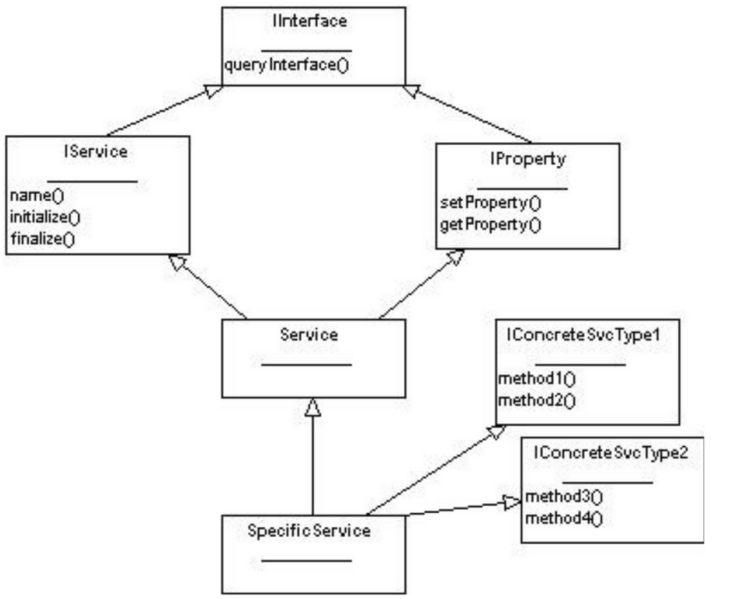


Рисунок 7 — Схема создания пользовательского сервиса.

3.4 Инструменты (Tools)

Алгоритмы могут использовать сервисы в ходе своей работы. При этом сервис создается в единичном экземпляре и может использоваться сразу несколькими алгоритмами или другими сервисами. Это неизбежно ведет к отсутствию возможности приватной настройки под каждого конкретного клиента. Также иногда хочется иметь возможность конфигурировать код, исполняемый алгоритмом, динамически. Логично выделить такой конфигурируемый код в отдельную сущность. Тогда эту сущность также можно будет сделать динамически конфигурируемой и снабжать каждого клиента своим собственным экземпляром. Такой сущностью в Gaudi являются *инструменты (Tools)*.

Если приватное конфигурирование инструмента не является необходимым и одним и тем же экземпляром могут пользоваться несколько алгоритмов или сервисов, то такой инструмент является *разделяемым*. В противном случае он является *приватным*. Различить, является инструмент приватным или разделяемым, можно по типу класса-родителя для данного инструмента: у разделяемых родителем является сервис ToolSvc, у приватных - конкретные алгоритмы либо же другие сервисы.

Сервис ToolSvc отвечает за создание инструментов и управление ими. Алгоритм запрашивает необходимые ему инструмент у ToolSvc, указывая, запрашивает ли он приватный экземпляр, путем объявления себя родителем для запрашиваемого инструмента. Алгоритмы, сервисы и другие инструменты могут объявлять себя родителями.

Базовым классом для создания собственных инструментов является AlgTool. При этом у разрабатываемого инструмента должен быть также и пользовательский интерфейс с предоставляемым им функционалом. Иерархия классов для разработки инструментов представлена на рисунке 8.

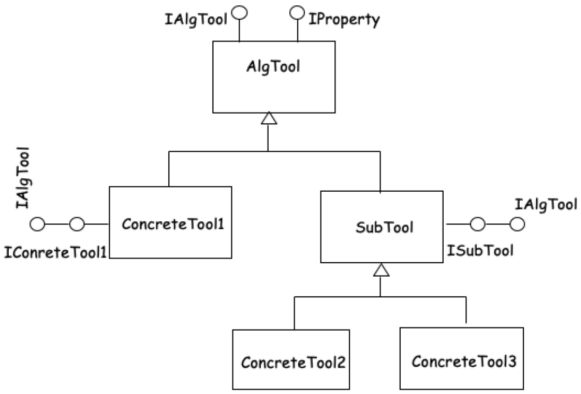


Рисунок 8 — Иерархия классов для разработки инструмента.

ToolSvc управляет инструментами. Он проверяет, доступен ли данный инструмент, и создает необходимый экземпляр после проверки, если он еще не существует. Если экземпляр инструмента существует, ToolSvc не будет создавать новый идентичный, а передаст алгоритму существующий экземпляр. Инструменты создаются по принципу “первого запроса”: первый алгоритм, запрашивающий инструмент, инициирует его создание. Взаимосвязь между алгоритмом, ToolSvc и инструментами показана на рисунке 9.

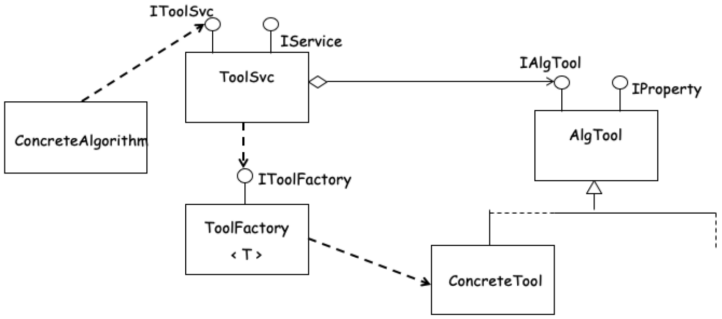


Рисунок 9 — Взаимодействие с ToolSvc.

3.5 Конфигурирование приложения. Job opotions

Под понятием *Job* имеется в виду запуск программы в определенной конфигурации на определенных входных данных. Для того чтобы сконфигурировать *Job*, в Gaudi предусмотрен механизм *JobOptions* файлов, представляющих собой набор команд, интерпретируемых Gaudi. На языке этих команд описывается последовательность алгоритмов, их параметры, используемы сервисы, входные данные и многое другое.

Однако большинство современных экспериментов использует другой подход. Он подразумевает конфигурирование задач с помощью скриптов на языке *Python*.

Ранее было сказано, что компоненты в Gaudi являются динамически конфигурируемыми. Для этого они должны быть отнаследованы от интерфейса IProperty и базового класса PropertyHolder. Именно через задание Properties производится конфигурирование приложения.

3.6 Контроль и планирование

Работа приложения начинается с создания экземпляра ApplicationMgr. ApplicationMgr отвечает за создание и инициализацию минимального набора необходимых сервисов, прежде чем управление будет передано другим компонентам (см. рисунок 10). EventLoopMgr отвечает за управление циклом обработки событий и планирование алгоритмов. Существует также возможность предоставить полный контроль над приложением компоненту, реализующему интерфейс IRunable. Это необходимо для интерактивных приложений, таких как отображение событий, интерактивный анализ и т.д.

Основными сервисами, которые ApplicationMgr необходимо создать, являются MessageSvc и JobOptionsSvc.

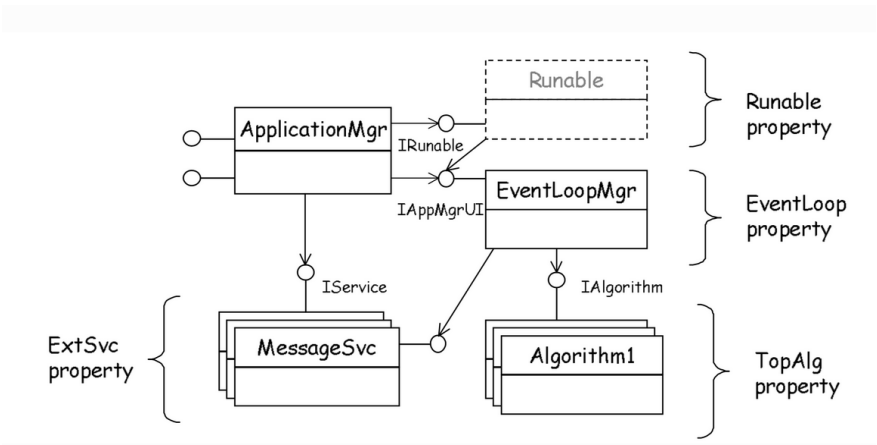


Рисунок 10 — Контроль и планирование в Gaudi.

3.7 Работа Gaudi-приложения

Общая схема работы приложения, написанного на базе Gaudi, представлена на рисунке 11:

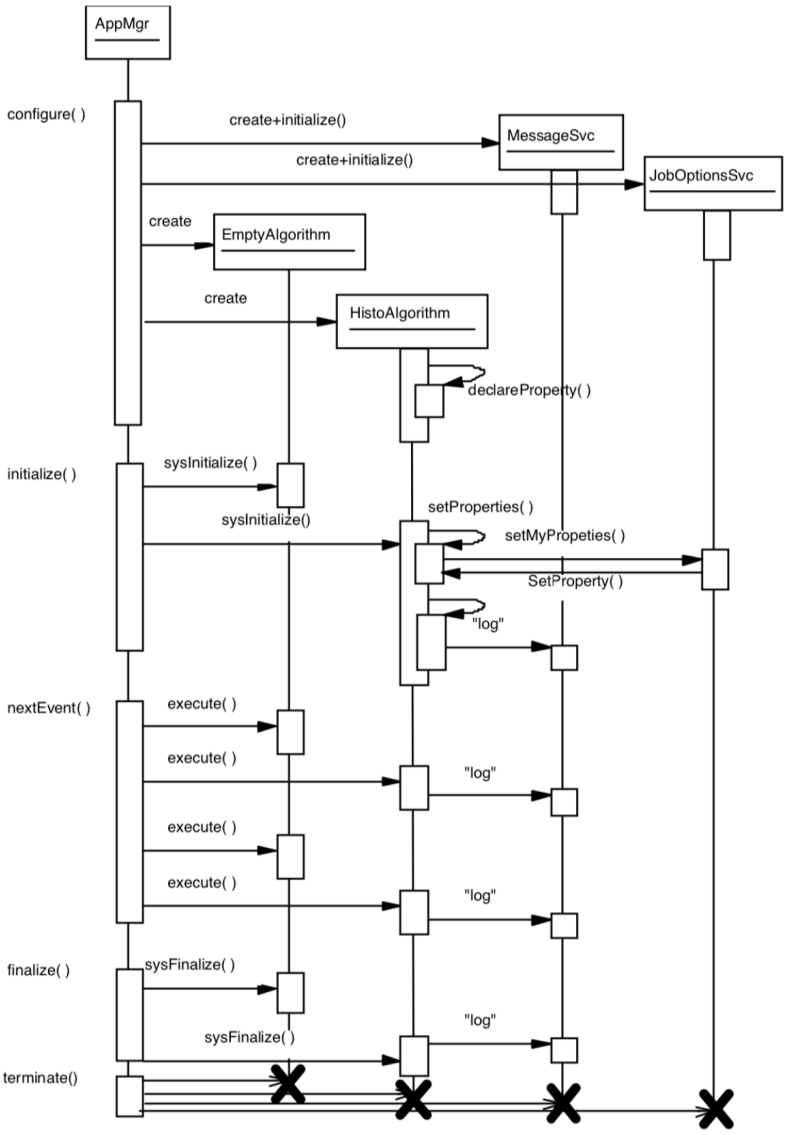


Рисунок 11 — Общая схема работы Gaudi-приложения

Порядок работы следующий:

- Application manager создает и инициализирует необходимые сервисы;
- создаются алгоритмы, указанные в JobOptions;
- устанавливаются свойства алгоритмов;
- Application manager начинает цикл обработки событий. Для каждого события вызываются алгоритмы в установленном порядке;
- по завершении цикла обработки событий алгоритмы завершаются;
- сервисы завершаются;
- освобождаются все ресурсы, программа завершается;

3.8 Многопоточность в Gaudi: GaudiHive

Описанная выше архитектура и порядок работы приложения - однопоточные. GaudiHive [10],[11] является расширением Gaudi, предназначенным для повышения производительности путем использования многопоточности.

В своей основе GaudiHive использует концепцию параллельной обработки задач (task parallelism, не путать task и job). Здесь задачей является выполнение алгоритма в контексте обработки какого-либо события. Зависимости этих задач друг от друга могут быть выражены в виде направленного ациклического графа (DAG), сформированного исходя из ввода-вывода алгоритмов. Пример такой зависимости представлен на рисунке 12.

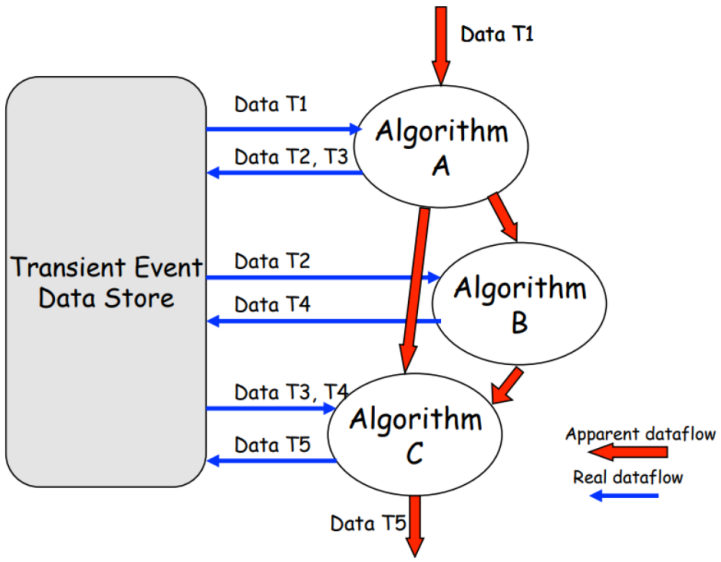


Рисунок 12 — Поток данных между алгоритмами, определяющий порядок выполнения задач.

GaudiHive планирует запуск задач исходя из доступности данных и ресурсов (т.е. свободных экземпляров алгоритмов для обработки). На практике это достигается следующим образом. Центральные элементы показаны на рисунке 13 и включают в себя специальный параллельный планировщик (Scheduler) и потокобезопасное хранилище (Whiteboard). Все алгоритмы должны объявлять свои входные и выходные данные на этапе инициализации. Если нет входных или выходных данных, то ничего не нужно делать.

Whiteboard - это менеджер хранилищ, каждое из которых является аналогом TES из однопоточной версии и ассоциировано с конкретным обрабатываемым событием. Как только в Whiteboard появляются новые данные, планировщик проверяет, есть ли в пуле алгоритмов экземпляры, зависимые от появившихся данных. В пуле может быть более одного экземпляра данного типа алгоритма. Затем из пула алгоритмов запрашиваются конкретный свободный экземпляр алгоритма, все входные зависимости которого имеются в наличии. Полученный экземпляр оформляется в задание и отправляется в пул потоков. Как только выполнение задания завершается, экземпляр снова передается в пул алгоритмов.

Таким образом, GaudiHive позволяет распараллеливать обработку событий, а также запускать несколько независимых алгоритмов параллельно в рамках обработки одного события.

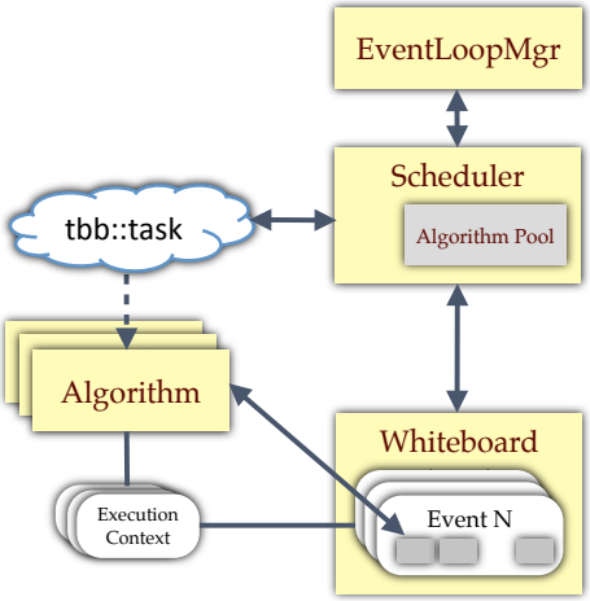


Рисунок 13 — Основные компоненты GaudiHive.

Событие считается обработанным, как только выполняются все вершины DAG (все алгоритмы). Но алгоритмы хранятся в пуле алгоритмов, они не связаны ни с каким конкретным событием. Для хранения информации о статусе обработки конкретного события используется концепция *Event Slots*.

Класс EventSlot можно рассматривать как словарь, ключами в котором являются алгоритмы, а значениями - их статусы выполнения. В разных слотах один и тот же алгоритм может быть помечен как имеющий разные состояния. Каждый слот ассоциирован с собственным хранилищем TES в Whiteboard. На рисунке 14 показан пример моментального снимка работы GaudiHive. Каждый цвет соответствует различному состоянию выполнения алгоритма.

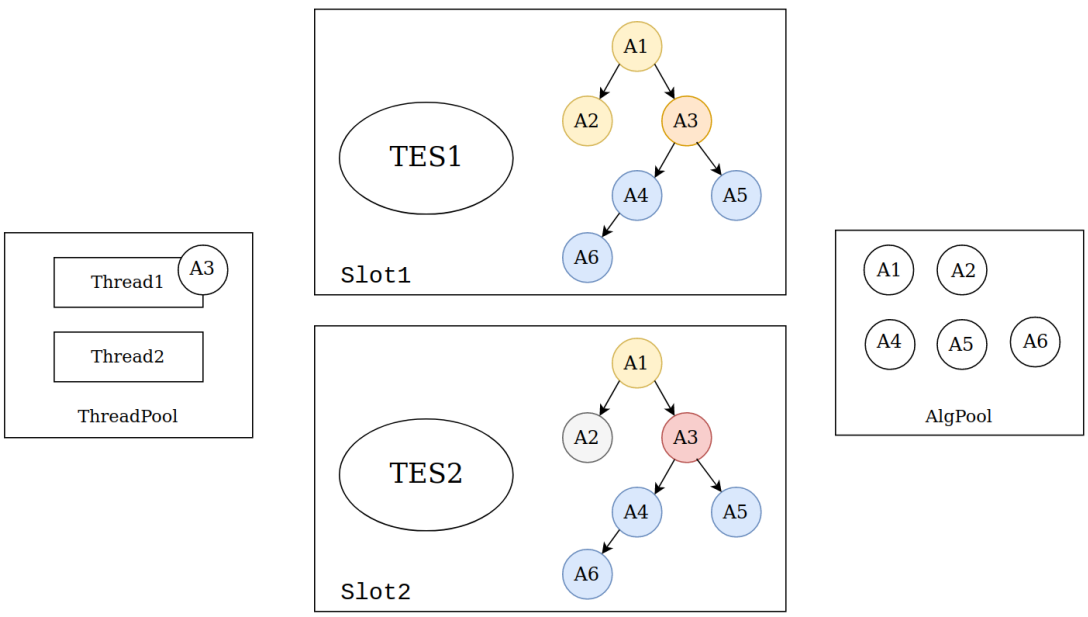


Рисунок 14 — Моментальный снимок GaudiHive в процессе работы.

4 Gaudi в SPD

В рамках разработки фреймворка на основе Gaudi было принято решение идти в соответствии с циклом жизни данных, то есть в порядке генерация-моделирование-реконструкция.

4.1 Интеграция Pythia8

Pythia8 [5] является универсальным Монте-Карло генератором событий для физики высоких энергий. Задачей генератора является создание коллекции выходных частиц, формирующих событие, в соответствии с некоторой физической моделью, начальными частицами и их характеристиками.

Задать начальные параметры Pythia8 можно двумя способами: построчно вызывать pythia.readString(...) или же один раз считать конфигурационный файл с помощью pythia.readFile(...). Второй способ выбирается в качестве основного, так как не требует перекомпилляции проекта, а также позволяет сохранить настройки генератора, которые в дальнейшем могут понадобиться.

Сгенерированные события необходимо хранить на диске. В некоторых случаях можно формировать комплексные задачи и одновременно с записью сразу же передавать сформированные события следующему в цепочке алгоритму моделирования. В любом случае необходимо учесть, что сгенерированные события должны быть представлены в некотором универсальном виде. В качестве такого представления выбирается HepMC3. Pythia8 оснащена встроенным конвертером, переводящим внутреннее представление события в HepMC3.

Генераторы событий, в соответствии с ходом их работы, в контексте Gaudi логично представить в виде алгоритмов. В таком случае генерация одного события будет соответствовать вызову метода *execute()* соответствующего генератору алгоритма. Общая схема организации классов для интеграции Pythia8 представлена на рисунке 15.

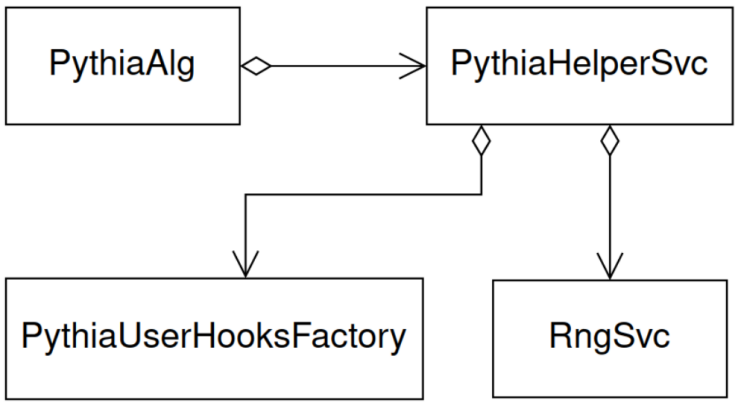


Рисунок 15 — Схема интеграции Pythia8.

Такой дизайн позволяет использовать многопоточность по аналогии с тем, как это было сделано в классе *PythiaParallel*, поставляемым разработчиками библиотеки Pythia8. В *PythiaParallel* создается пул потоков, каждому из которых соответствует собственный объект класса Pythia8. Эти объекты создаются путем клонирования вспомогательного объекта pythiaHelper класса Pythia8, который сам в генерации участия не принимает, а служит лишь для сбора статистики с прочих экземпляров, для которых он выступает в роли прототипа.

В нашем случае такой прототип хранится в сервисе PythiaHelperSvc. Настройка этого прототипа производится через задание свойств самого сервиса в JobOptions. PythiaHelperSvc выступает в роли фабрики для создания экземпляров Pythia8 на основе хранимого им прототипа. Указатели на все созданные экземпляры сохраняются в сервисе для сбора статистики, сервис же и отвечает за их уничтожение. Алгоритмы класса PythiaAlg являются клиентами PythiaHelperSvc. В методе *initialize()* они запрашивают экземпляр Pythia8.

Если Pythia8 используется совместно с другими библиотеками, то необходимо, чтобы во всех объектах, так или иначе ответственных за генерацию чего-либо, использовались одни и те же генераторы случайных чисел. Для этого вводится сервис RngSvc, выступающий в роли фабрики генераторов случайных чисел. Создаваемые в PythiaHelperSvc объекты Pythia8 снабжаются такими генераторами.

Также библиотека Pythia8 предоставляет пользователю возможность изменять стандартное поведение Pythia8 через механизм хуков. Созданный хук должен быть наследником класса Pythia8::UserHooks. Для динамического создания хуков используется класс PythiaUserHooksFactory, в котором зарегистрированы все возможные для создания виды хуков. Набор желаемых к использованию хуков передается в PythiaHelperSvc через JobOptions.

В разработанном дизайне есть одно ключевое отличие от *PythiaParallel*: в случае GaudiHive невозможно заранее знать, в какой поток попадет данный генератор, следовательно, невозможно назначить ему строго определенное число событий, которое он должен сгенерировать. Это ведет к потере детерминированности результата, что, скорее всего, не является желательным сторонним эффектом. Упрощенный пример потери детерминированности для случая двух потоков приведен на рисунке 16.

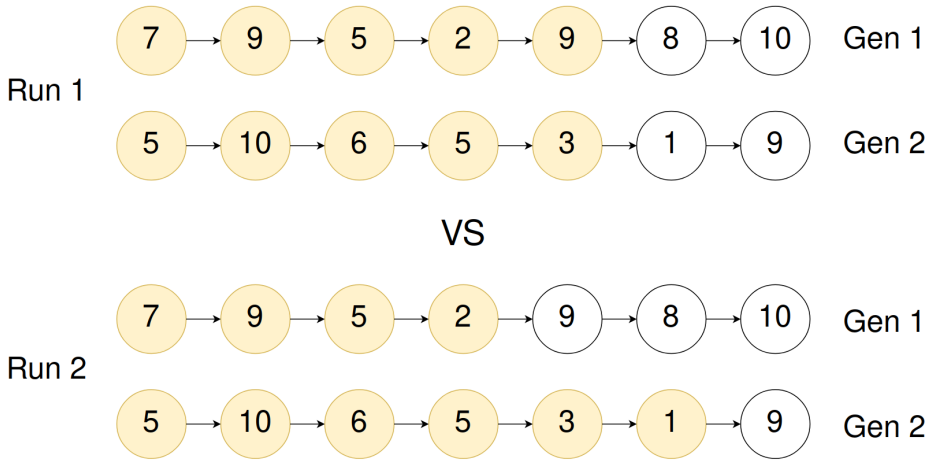


Рисунок 16 — Потеря детерминированности при многопоточной генерации за счет того, что заранее неизвестно, сколько раз каждый генератор будет вызван. В Run1 и Run2 получаются разные наборы чисел.

Устранить возникшую неопределенность возможно, если для каждого генерируемого события будет строго определенный seed для генератора. Рассчитывать этот seed можно на основе номера события и общего для данного запуска seed, передаваемого через JobOptions.

4.2 HepMC3

Библиотека HepMC3 [12] предназначена для хранения событий, создаваемых генераторами. Запись HepMC3 состоит из двух частей. Первая хранит информацию о параметрах генератора, вторая содержит вершины и ассоциированные с ними частицы. Частицы без конечной вершины считаются финальными.

Записи представлены объектами класса GenEvent. GenEvent может быть сериализован в простую структуру GenEventData, которую легко записать на диск. Для записи и чтения используются наследники классов HepMC3::Reader и HepMC3::Writer. Библиотека поддерживает несколько форматов записи. Наиболее производительным и экономным с точки зрения используемой памяти является формат ROOTTree - бинарный формат, основанный на использовании TTree из пакета ROOT [13]. Поддерживается также запись и в простые текстовые ASCII файлы.

Для записи HepMC3 в файлы разработан потокобезопасный сервис HepMCWriter. Он держит очередь запросов на запись, запросы могут поступать из разных потоков. Запросы обрабатываются в рабочем потоке, принадлежащем сервису.

4.3 Интеграция GeoModel

GeoModel [9]- это библиотека с минимальным числом зависимостей, предназначенная для описания детектора и хранения этого описания в базе данных. В рамках GeoModel объекты детектора хранятся в виде дерева объектов, корнем которого является так называемый мировой объем. Таким образом, принцип представления геометрии полностью аналогичен тому, как это реализовано в Geant4. Библиотека GeoModel снабжена конверторами, переводящими представление геометрии в формат, используемый в Geant4.

Так как описание геометрии может потребоваться многим компонентам фреймворка, то внутри Gaudi GeoModel реализуется в виде сервиса. В рамках проделанной работы был разработан соответствующий сервис GeoModelSvc, основными задачами которого являются:

- Обеспечение подключения к базе данных с описанием геометрии, проверка корректности подключения;
- Выгрузка элементов геометрии из базы данных (по первому запросу), создание дерева объектов;
- Предоставление мирового объема по запросу;

GeoModelSvc представляет собой фасад, изолирующий классы GeoModel от пользователя. На данном этапе также можно предвидеть, что в дальнейшем (например, на этапе реконструкции) может потребоваться не все дерево объемов, а лишь какая-то конкретная подсистема детектора. С целью предоставления такого функционала был разработан соответствующий дизайн, представленный на рисунке 17. Инструменты, реализующие интерфейс IDetGeoTool, являются клиентами GeoModelSvc и на его основе добывают параметры подсистемы, к которой они относятся.

Также возможно на определенном этапе придется создать более абстрактную сущность для получения геометрии, не привязанную к конкретной библиотеке.

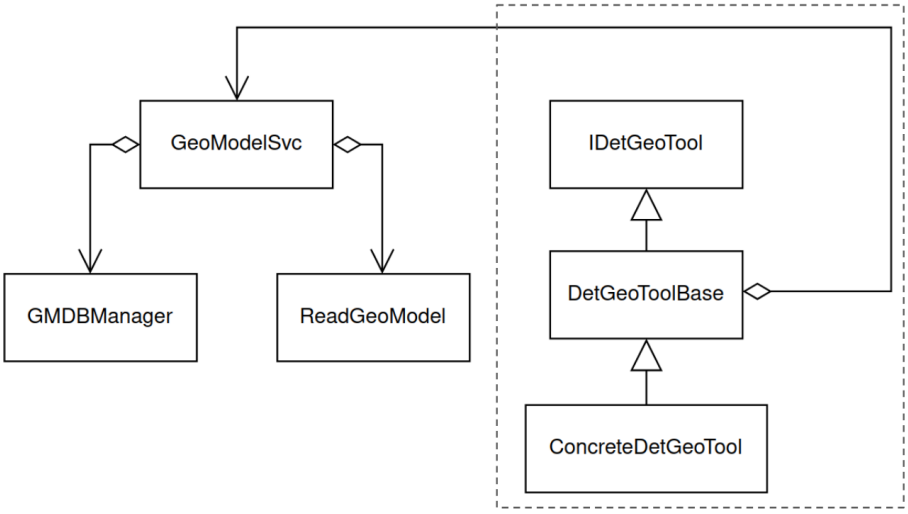


Рисунок 17 — Схема интеграции GeoModel.

4.4 Интеграция Geant4

Geant4 [8] предназначен для моделирования прохождения частиц через вещество детектора. Работой Geant4-приложения управляет объект класса G4RunManager. Он определяет ход выполнения программы и управляет циклом моделирования событий. G4RunManager также отвечает за управление процедурами инициализации, определенными в методах Initialization-классов, предоставляемых пользователем. С их помощью G4RunManager получает всю информацию, необходимую для создания и запуска симуляции, включая конструкцию детектора, моделируемые частицы и ассоциированные с ними процессы, правила генерации первичных частиц и т.д. К Initialization-классам относятся:

- *G4VUserDetectorConstruction* - предоставляет информацию о конструкции детектора;
- *G4VUserPhysicsList* - предоставляет информацию об учитываемых в моделировании физических процессах;
- *G4VUserActionInitialization* - предоставляет Action-классы;

Initialization-классы используются для инициализации всего процесса моделирования, в то время как Action-классы вызываются непосредственно в ходе моделирования на строго определенных этапах и могут использоваться как для сбора информации, так и для влияния на сам процесс моделирования. Общая схема пользовательских классов в Geant4 представлена на рисунке 18. К Action-классам относятся:

- *G4UserRunAction*
- *G4UserEventAction*
- *G4UserStackingAction*
- *G4UserTrackingAction*
- *G4UserSteppingAction*

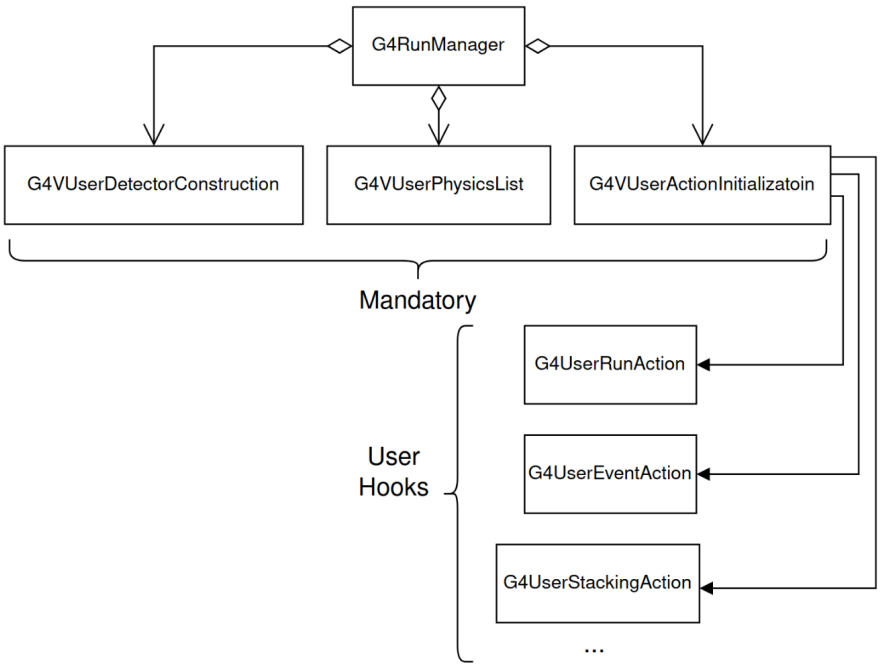


Рисунок 18 — Схема пользовательских классов в Geant4.

В контексте интеграции Geant4 в Gaudi необходимо разработать набор классов, которые будут представлять собой каркас для инкапсуляции и конфигурирования Geant4-частей внутри Gaudi.

Каждый из Initialization-классов предлагается вынести в отдельный сервис, как это показано на рисунке 19.

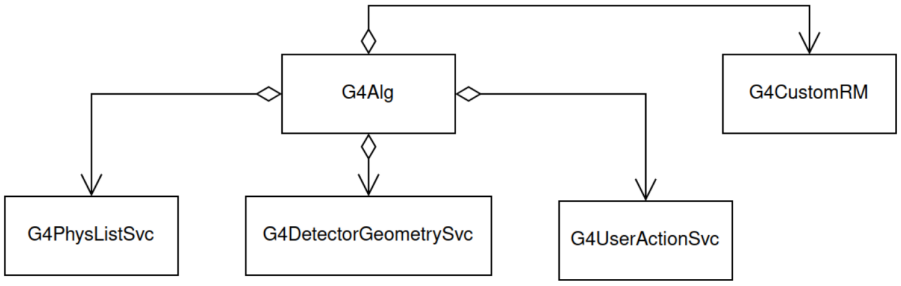


Рисунок 19 — Выделение Initialization-классов в сервисы в рамках Gaudi.

Сервисы, являясь частью инфраструктуры Gaudi, будут обладать всеми её преимуществами, главным среди которых является возможность динамического конфигурирования. Таким образом, без необходимости в пересборке, пользователи будут способны через механизм JobOptions выбирать конкретные реализации того или иного компонента из Geant4. С точки же зрения разработки прикладного кода, конечному пользователю будет необходимо реализовывать интерфейсы, относящиеся к определенному сервису. Рассмотрим предлагаемое устройство всех трех сервисов.

Устройство G4PhysListSvc довольно простое, оно представлено на рисунке 20 и следует идее определения физических процессов в Geant4.

Сервис содержит объекты, реализующие два типа интерфейсов. Объект с интерфейсом IG4PhysListTool задается в единичном экземпляре, он служит в качестве

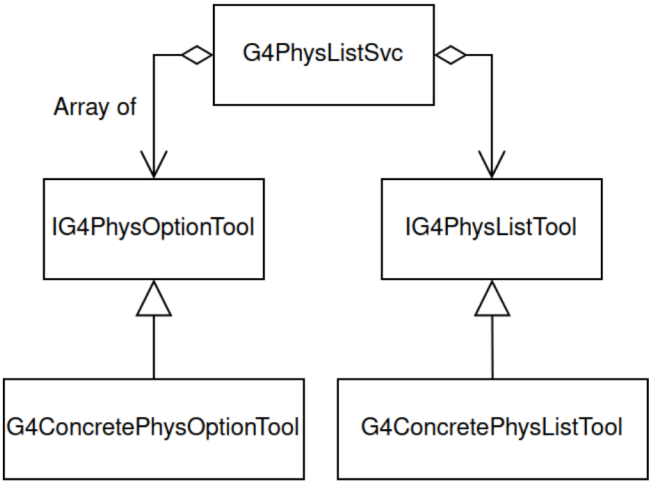


Рисунок 20 — Устройство G4PhysListSvc.

фабрики для PhysicsList. PhysicsList можно расширить путем задания опций, фабриками которых будут объекты с интерфейсом IG4PhysOptionTool.

Устройство G4DetectorGeometrySvc сложнее, оно представлено на рисунке 21.

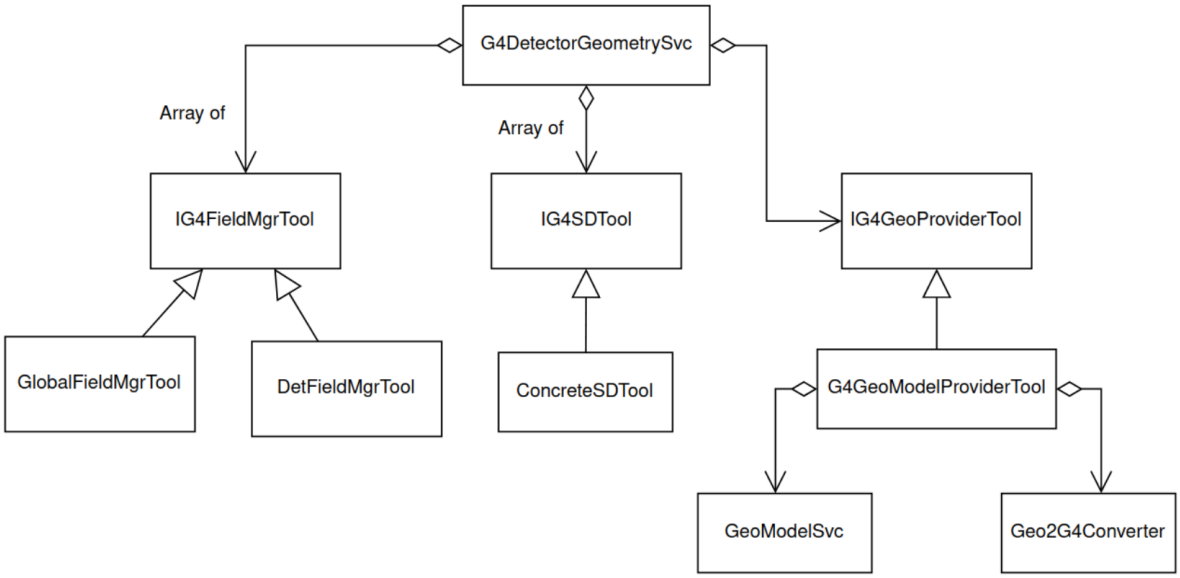


Рисунок 21 — Устройство G4DetectorGeometrySvc.

IG4GeoProviderTool отвечает за конструирование дерева объемов в Geant4. Конкретной реализацией является, например, G4GeoModelProviderTool, работающий в кооперации с GeoModelSvc.

IG4SDTool является фабрикой чувствительных детекторов. То есть разработав Geant4-класс для чувствительного детектора, в Gaudi также необходимо будет предоставить соответствующий ConcreteSDTool.

Для управления магнитными полями используются IG4FieldTool. Поля могут быть как локальными для какого-то участка, так и глобальными для всего детектора.

Наконец, устройство G4UserActionSvc представлено на рисунке 22. G4UserActionSvc держит MasterTool для каждого вида Action-класса. MasterTool, в свою очередь держит массив конкретных Tool для данного Action-класса.

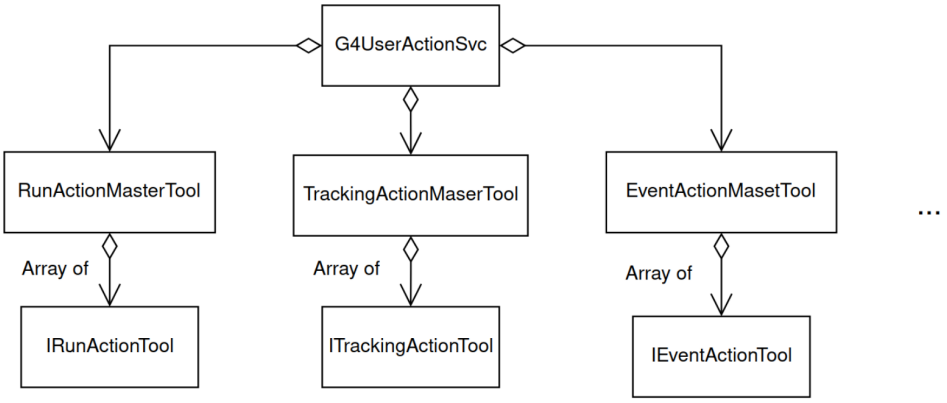


Рисунок 22 — Устройство G4UserActionSvc.

5 Заключение и дальнейшие планы

Данная работа посвящена разработке offline программного обеспечения для эксперимента SPD на базе платформы Gaudi. Были рассмотрены основные компоненты и архитектура Gaudi, многопоточная архитектура GaudiHive, а также основные этапы жизненного цикла данных в физическом эксперименте. Каждый из этапов требует использования специализированных библиотек, которые необходимо интегрировать в общий фреймворк.

Результатами проделанной работы стали:

- интеграция генератора Pythia8 в инфраструктуру Gaudi в виде соответсвующего набора из алгоритма, сервиса и фабрики хуков. В дальнейшем, если потребуются другие генераторы, они могут быть интегрированы аналогичным образом;
- обеспечение записи сгенерированных событий на диск в формате HepMC в виде соответствующего потокобезопасного сервиса;
- интеграция библиотеки GeoModel в инфраструктуру Gaudi в виде соответствующего сервиса;
- начат процесс интеграции библиотеки Geant4 в инфраструктуру Gaudi. Разработан дизайн классов, представляющих собой каркас для работы с Geant4 внутри Gaudi. Текущий дизайн является однопоточным;

Дальнейшие планы:

- завершить интеграцию Geant4 с однопоточным дизайном. Разработать и реализовать многопоточный дизайн;
- приступить к разработке прикладных алгоритмов реконструкции;
- осуществить интеграцию Gaudi фреймворка с другими компонентами ПО эксперимента SPD;

Список использованных источников

1. The FairRoot framework / M. Al-Turany [и др.] // Journal of Physics: Conference Series. — 2012. — Дек. — Т. 396, № 2. — С. 022001.
2. *Mato P.* GAUDI-Architecture design document. — 1998. — Нояб.
3. *Sivers D. W.* Single Spin Production Asymmetries from the Hard Scattering of Point-Like Constituents // Phys. Rev. D. — 1990. — Т. 41. — С. 83.
4. On the physics potential to study the gluon content of proton and deuteron at NICA SPD / A. Arbuzov [и др.] // Prog. Part. Nucl. Phys. — 2021. — Т. 119. — С. 103858. — arXiv: 2011.15005 [hep-ex].
5. An introduction to PYTHIA 8.2 / T. Sjöstrand [и др.] // Comput. Phys. Commun. — 2015. — Т. 191. — С. 159—177. — arXiv: 1410.3012 [hep-ph].
6. *Andersson B.*, *Gustafson G.*, *Nilsson-Almqvist B.* A model for low-pT hadronic reactions with generalizations to hadron-nucleus and nucleus-nucleus collisions // Nuclear Physics B. — 1987. — Т. 281, № 1. — С. 289—309. — ISSN 0550-3213.
7. Microscopic models for ultrarelativistic heavy ion collisions / S. A. Bass [и др.] // Prog. Part. Nucl. Phys. — 1998. — Т. 41. — С. 255—369. — arXiv: nucl-th/9803035.
8. GEANT4–a simulation toolkit / S. Agostinelli [и др.] // Nucl. Instrum. Meth. A. — 2003. — Т. 506. — С. 250—303.
9. Going standalone and platform-independent, an example from recent work on the ATLAS Detector Description and interactive data visualization / S. A. Merkt [и др.] // European Physical Journal Web of Conferences. Т. 214. — 07.2019. — С. 02035. — (European Physical Journal Web of Conferences).
10. *Hegner B.*, *Mato P.*, *Piparo D.* Evolving LHC data processing frameworks for efficient exploitation of new CPU architectures. — 2012.
11. Introducing concurrency in the Gaudi data processing framework / M. Clemencic [и др.] // Journal of Physics: Conference Series. — 2014. — Т. 513, № 2.
12. The HepMC3 event record library for Monte Carlo event generators / A. Buckley [и др.] // Comput. Phys. Commun. — 2021. — Т. 260. — С. 107310. — arXiv: 1912.08005 [hep-ph].
13. ROOT: A C++ framework for petabyte data storage, statistical analysis and visualization / I. Antcheva [и др.] // Comput. Phys. Commun. — 2009. — Т. 180. — С. 2499—2512. — arXiv: 1508.07749 [physics.data-an].